Improving the Reliability of Pervasive Computing Applications By Continuous Checking of Sensor Readings

Adrien Carteron, Charles Consel, Nic Volanschi

Improving the Reliability of Pervasive Computing Applications By Continuous Checking of Sensor Readings

Adrien Carteron
Inria Bordeaux, France
Email: adrien.carteron@inria.fr

Charles Consel
Bordeaux Institute of Technology, France
Email: charles.consel@inria.fr

Nic Volanschi
Inria Bordeaux, France
Email: eugene.volanschi@inria.fr

***Abstract*—This paper shows that context-aware applications commonly make implicit assumptions about a sensor infrastructure. Because context-awareness critically relies on these assumptions, the developer typically need to ensure their validity by encoding them in the application code, polluting it with non-functional concerns. This defensive programming approach can be avoided by formulating these assumptions aside from the application, thus factorizing them as an explicit model of the sensor infrastructure. This model can be expressed as a set of rules and can be checked *automatically* and *continuously* to ensure the reliability of a sensor infrastructure, both at installation time and during normal functioning. The usefulness of our approach is demonstrated in the domain of assisted living for seniors. We applied it to sensor data collected in the context of a 9-month field study of an assisted living platform, deployed at the home of 24 seniors. We show that several kinds of sensor malfunctions could have been identified upon their occurrence, thanks for our continuous checking, and resolved.**

***Keywords*-context-aware systems; smart homes; pervasive computing; reliability; sensor networks; assistive applications**

I. Introduction

As pervasive computing applications get intertwined with users' daily activities, context awareness becomes a key enabling feature to make these applications acceptable to users. For example, in a smart home, context awareness allows an application to remind the user of an activity [1], *only* when it has been forgotten; it triggers an alert for an open door, *only* when nobody is in the surroundings; it calls upon the user for a hot cooker, *only* when it has been unattended for a while. To pervade the user's life, the context-awareness of activity-supporting applications must be reliable. This property is essential to minimize false positives in detecting situations that require the user attention. Indeed, inaccurate context awareness leads to notification fatigue in users, preventing technology adoption.

The reliability of a context-aware, pervasive computing application is defined by the reliability of the system as a whole [2]. In pervasive computing, the system mainly consists of the software and hardware layers. Software reliability has been studied for the pervasive computing domain; simulation-based approaches have been proposed (*e.g.,* [3]) to test the software extensively before deploying it. These approaches allow context-aware scenarios of applications to be tested in-vitro, simulating interactions of the user with the environment (*e.g.,* an open door, a presence in a room, a switched-on appliance). Despite the rigor and thoroughness applied to this testing phase, the resulting software reliability holds to the extent that the underlying hardware layer is itself reliable.

The reliability of the hardware layer of a pervasive computing system depends on a number of factors. At a unitary level, it depends on the reliability of a sensor, which could at first amount to whether the sensor is functioning properly. But in fact, when it is context aware, an application relies on a number of requirements regarding sensors [2]. Conceptually, a sensor is viewed as playing a *role* in the application logic: it measures the environment, including the interactions of the user with their environment. As such, the sensor is assumed to be placed at an appropriate location to measure interactions accurately (direction, frequency, non-interference, *etc.*) [4]. When determining a context depends on a combination of interactions, the developer builds an implicit model of the roles to be played by a combination of sensors [5]. For example, a contact sensor for the entrance door can be conceptually coupled with a motion detector, covering the entrance area. An application can use these dependent sensors as follows: if the entrance door is open, but the presence of the user is detected nearby, then no alert should be delivered. If this implicit dependency model is violated because of a faulty motion detector, the context awareness of the application is compromised, resulting in notifications erroneously alerting about the entrance door left open and unattended, even when the user stands in the entrance area.

To address reliability at the sensor level, a developer typically introduces code to test whether their implicit model is valid. This strategy consists of translating *implicit rules* into conditional statements dispatched throughout the application code. These rules contribute to ensure that sensor readings are in conformance with the implicit model. For example, let us assume that a kitchen is equipped with contact sensors for specific cabinet doors and a motion detector. Then,

This work was partially funded by Région Aquitaine, Limousin, Poitou-Charentes.

a cabinet door should not be detected as being opened without first detecting a presence in the kitchen. If that situation occurs, it means that the implicit model is violated. A number of reasons could then be invoked (placement, malfunction) and should lead to an intervention to solve the problem. Detecting these situations is a key to the reliability of context-aware pervasive computing applications in the wild [6].

This paper addresses the challenge of making reliable the detection of user activities defined by interactions of user with their environment. This range of context awareness is of the utmost importance for a range of applications supporting daily life activities of a user in their home. Our approach consists of making explicit the *model of roles* played by sensors. This model is defined by declaring rules that contribute to ensuring that sensors function in conformance with their expected roles. These *conformance rules* take the form of relations that are checked by a Prolog program. This program runs alongside the pervasive computing system; it intercepts sensor readings and continually checks whether they conform to the model of sensor roles, raising an error if they do not.

We implemented our approach with a complete system combining 1) a pervasive computing platform, 2) sensors and actuators for the home, and 3) a Prolog-based verification layer. This implementation has been applied to the domain of assisted living for older adults. We used deployments made in the home of older adults to validate our tool. Specifically, we demonstrated that our tool can detect a number of anomalies in deployments, ranging from misplacement to obstruction or fall.

Our approach has many benefits, impacting all the stakeholders of a smart home. Programmers can make their application requirements on sensors explicit as they develop their application, while preventing their code from being polluted with conditional statements. This achieves a *separation of concerns* that contributes to ease the development of pervasive computing applications. Also, when a sensor is shared among different applications, its declared role can be matched against a new application requirement. Furthermore, the model of sensor roles can be shared with the individual installing the sensors. This could be done by providing them with an appropriate representation for the conformance rules to guide in the placement of the sensors.

This paper makes the following contributions.

- We propose an approach to improving the reliability of context-aware applications dedicated to activity monitoring.
- A concept of *model of sensor roles* is introduced to capture application requirements concerning sensors, and continuously checked alongside running applications.
- This model of sensor roles can be *factorized* across applications, *shared* among stakeholders, and *leveraged* for platform evolution.
- We present a tool that implements our approach in an assisted living platform.
- We validate our tool on real deployments in the home of older adults, demonstrating the effectiveness of our approach.

The paper is organized as follows. Section II describes a case study used for introducing and illustrating our approach. Section III analyzes the application needs in terms of sensing abstractions and their implicit assumptions about these abstractions. Section IV proposes a model for making these assumptions explicit and checkable automatically. Section V presents the architecture we propose for continuously checking the model conformance during execution, or offline. Section VI describes some experimental results validating our approach on data accumulated during an activity monitoring field study. Section VII further discusses some benefits of our approach. Section VIII relates our approach to other research works, and Section IX concludes.

II. Case study

To illustrate our approach, we present the monitoring of two kitchen activities as a case study. This case study is simple but complete, using several types of sensors to recognize different types of user interactions in the kitchen environment.

Activity monitoring is strongly dependent on the population targeted by the assistive applications [7]. For example, older adults may simply need to be reminded of daily tasks [8], whereas people with intellectual disabilities (*e.g.,* Down syndrome) may need to be monitored through key steps of a task [9].

To determine *which* activities to monitor and *how* to monitor them, we need expertise on the targeted population from such professionals as ergonomists and occupational therapists. In our example, an expert is able to decide which activities must be monitored in the kitchen. Starting from this set of activities, the expert then defines *which interactions* with the environment need to be detected for each activity of interest. Typically, the expert will ask users to mimic the steps they perform during an activity [8]. Once the key interactions related to each activity of interest have been identified, we need to define *which sensors* are relevant to measure these interactions. This phase has been studied by Beckmann *et al.* [6]. They present practical guidelines for installing sensors in the home and assess them with a field study.

Let us assume that the expert suggested to monitor the breakfast preparation to issue a reminder to the user when this activity is missed and to monitor the cooker to ensure it is safely operated.

A. Breakfast activity monitoring

To monitor the breakfast activity, the domain expert provides typical scenarios performed by users. One of them

involves a user (1) making coffee using an electric coffee maker, (2) taking a cup from a specific cabinet or the dishwasher, and (3) taking milk from the fridge. From this scenario, one can isolate the interactions to be recognized: (1) turning on the coffee maker, (2) opening a cabinet door, and (3) opening the fridge door. To detect these interactions, an electric sensor is used to measure electric consumption associated with the coffee maker: sensing electric consumption means that the coffee maker is turned on. Interactions with the fridge door and a cabinet door are both detected using contact sensors. For simplicity, we omit putting a contact sensor on the dishwasher and assume that the action of taking a cup may not be detected.

B. Cooker safety

Monitoring the cooker for safety depends primarily on detecting the cooker usage and the nearby user, keeping an eye on the cooking process. An electric sensor is used to detect whether the cooker is turned on. To determine whether the user is nearby, a motion sensor can be positioned to monitor presence within a strategic perimeter around the cooker (typically the kitchen).

C. Making explicit an implicit model

To summarize, the activities to be monitored in our case study rely on recognizing five interactions with the environment of the kitchen: opening a cabinet door, opening a fridge door, turning on a coffee maker, turning on a cooker, and detecting presence in the kitchen.

Let us now sketch, in the context of our case study, the last step of our proposed method: making explicit the implicit model that ensures the reliability of the detected interactions. We know that each interaction measured in the kitchen must be preceded by a presence detection. Conversely, if an interaction is measured without being preceded by a presence detection in the kitchen, the sensors involved are assumed to be malfunctioning. These rules contribute to ensuring that the sensor infrastructure functions in conformance with the model of roles.

III. Application requirements

The aim of activity-supporting applications is to provide monitoring and assistive services to the user. The services rely on a set of interaction measurements in the environment, as illustrated by our case study presentation.

However, there is often a gap between the raw data, delivered by the sensors, and the conceptual view of an environment interaction, required by the application developer. For example, a contact sensor produces boolean values, defining an open/close status. This status needs to be combined with the specific location (and/or identifier) of the sensor to make an interaction meaningful for the application. Furthermore, multiple sensors may need to be combined to detect a situation. For example, several motion detectors may be needed to detect presence in an L-shaped room.

Our approach aims to abstract over the physical layout of sensors needed to detect a conceptual situation. To do so, we introduce the notion of *roles* required by an application.

A. Role

From the application viewpoint, a role defines interaction information that can be directly used by the application logic; it consists of a type of interaction (*e.g.,* presence) and the location of the interaction (*e.g.,* the kitchen). From a physical viewpoint, a role defines requirements that must be fullfilled by one or more sensors to detect a given interaction. As a result, roles are placed between the application and the physical layout of sensors, as depicted in Fig. 1. To be in conformance with a role, the sensors must be layed out properly in terms of positioning and direction, for example. As illustrated previously, detecting presence in an L-shaped kitchen requires at least two motion sensors, appropriately directed so that each part of the kitchen be covered, without picking up motion in adjacent spaces (*e.g.,* the hallway outside the kitchen). As can be noticed, roles allow a separation of concerns between the application and the physical layout of sensors.

B. Semantics of roles

In fact, the semantics of a role amounts (1) to detecting an interaction with the environment, producing a value *true*, and (2) to detecting the end of this interaction, producing a value *false*.

A role is thus a function defined over time and ranging over boolean values. When an interaction is detected at a given time, it produces true, until it is no more detected, at which time is produces false.

In practice, note that to provide this timed event semantics to the application, the implementation of a role must filter out atypical sequences of sensor readings (noise). For example, the role layer of a contact sensor, when provided with two consecutive true values (open), will filter out the second one.

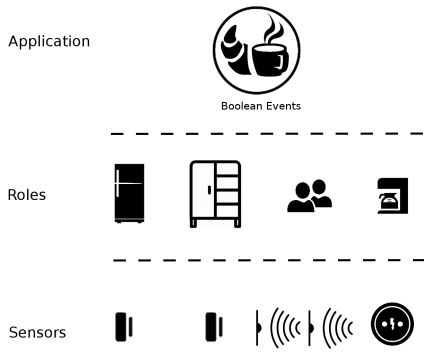

Figure 1. Application requirements and roles

IV. Infrastructure model

The sensors installed in a home form an infrastructure that supports the roles required by the deployed applications. The reliability of the context awareness of this infrastructure critically depends on the correctness of role implementations.This reliability goes beyond unitary tests for each role. To address the reliability of the sensing infrastructure, we propose to build a checkable model of this infrastructure. Not only does this model take into account individual roles, but it also considers their conformance with respect to a set of global rules about the sensor infrastructure.

A. Role events domain

The first step to make explicit the infrastructure model as a set of rules is to provide the domain of objects that rules act upon: the role events. A role event consists of three elements: (1) an interaction that occurred (2) at a given location, (3) during a specific period of time. First, let us examine the notion of period. It is defined as an interval bounded by two timestamps. That is,

$$Period = \mathbb{N}^2$$
$$For\ p \in Period, p =< t_1, t_2 >\ and\ t_1\ <\ t_2$$

A period can thus be seen as a set of increasing time values, from t_1 to t_2, with an increment of 1 second – finer granularity is not useful in practice. As such, we use the basic operations on sets to operate on periods, including $\subseteq, \supseteq$.

Second, a role event consists of an interaction that occurred during a period. The set of interactions is defined by $Inter$ (*e.g.,* Presence, Opening, Use). Last, a set of locations, Loc, specifies the locations of interest in the home (*e.g.,* Kitchen, Bathroom, Bedroom). Role events are thus defined as follows.

$$e\ \in\ Event = Inter \times Loc \times Period$$

As the infrastructure of sensors monitors the home, it produces a log of readings that is structured as a stream of role events, defined above. The log of role events is defined as $log\ \in\ Log = \mathscr{P}(Event)$

B. Formulating rules

Now that logs of role events have been defined and can be manipulated, let us focus on the rules of the infrastructure model. These rules are expressed as a set of logical formulas in the first-order predicate calculus. We introduce these rules by examining three examples from our case study.

Kitchen presence. We define a rule that makes explicit the dependency of the sensors in the kitchen. In essence, we want to express the fact that any detected interaction, which is not motion, must be surrounded by motion interaction. In doing so, we express the fact that the motion sensor of the kitchen encompasses any other interactions in the kitchen (*e.g.,* cabinet door, coffee maker). Once expressed, this semantics ensures the consistency of the sensor readings in the kitchen.

Our kitchen-presence rule takes a role event situated in the kitchen and a log; it is defined as follows.

$$\forall\ < i, Kitchen, p >\ \in\ Log, i \neq Presence\ \Rightarrow$$
$$\exists\ < Presence, Kitchen, p' >\ \in\ Log,\ p \subseteq p'$$

Left-open doors. We assume that for assisted living purposes, doors equipped with contact sensors must not remain open beyond a given period of time, noted MAX. Such a rule typically applies to the fridge door and the entrance door because they must not remain open for too long. The MAX time can vary depending on the user preferences and the door type.

This rule takes an opening role event and a log; it is defined as follows.

$$\forall\ < Opening, l, p >\ \in\ Log \Rightarrow \#p < MAX$$

Notice that a door left open may mean that the associated sensor is malfunctioning, or that the user has in fact forgotten this door and it is not monitored by an application triggering a safety notification. For example, the cabinet door in our case study is only monitored for reminding the user to prepare their meal, not for reminding them that it is left open.

No ubiquity. Some conformance rules can be specific to a given application area. For example, our research in pervasive computing is partly dedicated to independent living of single seniors. This situation can give rise to the following conformance rule: a presence role cannot be detected simultaneously in two locations. The no-ubiquity rule is defined as follows.

$$\forall\ < Presence, l, p >\ \in\ Log \Rightarrow$$
$$\nexists\ < Presence, l', p' >\ \in\ Log,\ l \neq l' \wedge\ p' \cap p \neq \emptyset$$

In practice, defining conformance rules provides detailed guidelines to install and position sensors in the physical world. For example, the kitchen-presence rule requires the presence to be recognized in the entire kitchen. Once a home is installed, the rules ensure the compliance of the installation to the model.

V. Architecture

In this section, we propose an architecture to *continuously* verify that an installation is in conformance with its infrastructure model. Then, we briefly describe our prototype implementation of this architecture, which has been used to validate our method experimentally.

A. Architecture

Globally, our proposed architecture consists of abstracting over raw sensor readings with the role layer, fueling both the relevant applications with high-level values, and the log of role events used by the conformance rules of the

infrastructure model. This architecture is depicted in Fig. 2. As can be noticed, role events are processed concurrently by the relevant applications and the log component. In doing so, the conformance rules can be executed on the fly to raise errors as they occur. Alternatively, the rules can be executed offline to diagnose problems when an operator is available for maintenance.

Beyond maintenance, logs of role events can also be very valuable for analysis purposes in the case of assisted living of seniors. Indeed, these logs allow to perform longitudinal analyses of user activities that can reveal aspects of cognitive decline due to aging. Such analyses will typically prompt professionals to adapt, remove or install new assistive applications to address the evolving needs of the users.

As well, application developers can also leverage logs of role events to adjust the application logic to the reality of user daily life. For example, one may adjust thresholds triggering notifications to prevent user fatigue. Also, a log of role events may be used by the developer to replay a sequence of interactions to debug an application that has crashed or behaved incorrectly.

B. Implementation

This section focuses on the components needed to implement the conformance rules. Our implementation revolves around Prolog because it allows to naturally express our rules and efficiently performs conformance checking.

Rules.

Conformance rules are implemented as Prolog predicates, as well as operators to manipulate role events. More precisely, *inter_e/2*, *locat_e/2*, and *period_e/2* allow to access the interaction, the location, and the period of an event, respectively, or to check their equality to a given constant. Operator *subset_e/2* checks whether the period of an event is included in the period of another event.

As depicted in Fig. 3, the Prolog predicate *presence_kitchen/1* checks the rule on the log of role events, represented as a list, and returns the list of events that violate the rule. A helper predicate *presence_kitchen/2* with two arguments is used for examining each event in the log while passing around the complete log. For each role event *E* in the log, which is not a presence and which happens in the kitchen, the helper predicate checks the rule using the *presence/2* predicate; if this fails, it adds the current event to the list of violating events. The *presence/2* predicate searches for a presence role event in the kitchen, whose period encompasses *E*, using operator *subset_e/2*; if no such event is found in the whole log, the rule fails. Note that the implementation given above is naive because it scans the complete event log for each event to be checked, so it has quadratic running time. The actual implementation searches a reduced log (by filtering only events in the kitchen) and is linearized (by considering each presence event in the kitchen and filtering out all the events included by it).

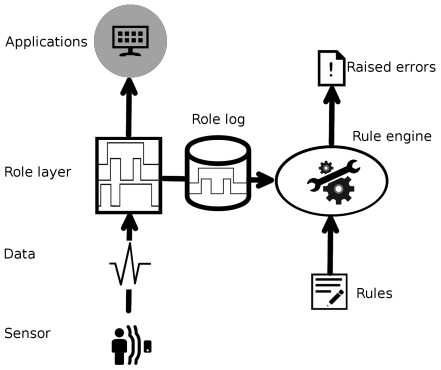

Figure 2. Architecture

Role parser. Because the pervasive computing platform used to validate our method did not provide the role abstract layer, we simulated this layer in our implementation of the architecture using a "role parser". This component builds a log of role events by processing the log of sensor readings and associated information (sensor type, location, state, and timestamp). The role parser is implemented as a C++ module.

Rule engine.

This module processes the log of role events produced by the role parser, and calls the Prolog interpreter to execute each rule. As a result, when a rule fails, the rule engine

```
% Checks rule on all the log. Takes the whole log.
% Returns list of violating events.
presence_kitchen(Log,LVE):-
    presence_kitchen(Log,Log,LVE).

% Checks rule on all the log (helper predicate).
% Takes remaining log to check and the whole log.
% Returns list of violating events.
presence_kitchen([],_,[]).
presence_kitchen([E|T],Log,[E|LVE]):-
    not inter_e(E, 'Presence'),
    locat_e(E,'Kitchen'),
    not presence(E,Log),!,
    presence_kitchen(T,Log,LVE).
presence_kitchen([_|T],Log,LVE):-
    presence_kitchen(T,Log,LVE).

% Checks rule on a kitchen non-presence event.
% Takes event to check and the remaining log.
presence(E, [E1|_]) :-
     period_e(E,P),
     inter_e(E1,'Presence'),
     locat_e(E1,'Kitchen'),
     period_e(E1,P1),
     subset_p(P,P1),!.
presence(E, [_|T]) :-
     presence(E,T).
```

Figure 3. presence_kitchen Prolog predicate

identifies the role events involved and provides a list of non-conformant role events along with the corresponding failing rules. This module is also implemented as a C++ module.

VI. Validation

We first briefly present the experiment that we used to collect real log data. We then defined conformance rules for a model of roles dedicated to assisted living of seniors. Finally, we applied the rules to the logs to assess their ability to detect conformance violations.

A. HomeAssist experiment

The HomeAssist project[1] aims to prolong independent living for single seniors in their home by providing them an assisted living platform with applications supporting their daily activities. Experts in occupational therapy and psychology and aging have defined which activities to monitor then and a set of environment interactions to be measured.

In this project, two kinds of applications were provided: (1) applications for monitoring daily activities and assist the user when they are missed, and (2) safety applications to secure the home (*e.g.,* entrance door left open). A field study was conducted by recruiting 24 older participants and deploying our assisted living platform for a period of 9 months. This present work uses the sensor data collected during the HomeAssist project to validate our approach.

B. Model

The sensor setting of HomeAssist allows to detect twelve points of interaction with the environment. Presence interactions are measured with motion detectors, opening interactions are measured with contact sensors, and electric appliance uses are measured with electric consumption sensors. The HomeAssist setting and the related roles are summarized in Table I.

For practical reasons, our approach was not put into practice when HomeAssist started. Instead, we retrofitted HomeAssist with our approach. As a result, our model was used to retroactively check the conformance of each participant's home by executing the rules on the logs accumulated during the experiment. In an ongoing field study, our work will be part of the deployed platform.

Room	Role	Sensor
Kitchen	Coffee maker in use	EM
	Cabinet door open	CS
	Fridge door open	CS
	Microwave in use	EM
	Presence	CS
Entrance	Door open	CS
	Presence	MD
Bathroom	Shower in use	MD
	Presence	MD
Bedroom	Dressing open	CS
	Bedside lamp in use	EM
	Presence	MD

EM = Electric Meter, CS = Contact Sensor, MD = Motion Detector.

Table I
HomeAssist roles

Considering the field study and our roles, we specified the conformance rules that made explicit the setting of our field study. Namely, participants lived alone (No ubiquity rule) and some interactions with the environment had to follow a typical pattern (Left-open doors). Other conformance rules were agnostic to the purpose of the study and could be generalized to any setting. This situation applies to the Presence inclusion rule and its refinements, Presence intersection and Presence requirement.

Let us now further examine these rules.

No ubiquity. Recall that this rule ensures that a presence role is not detected simultaneously in two locations. In practice, depending on the reactivity of the motion sensors used to implement presence detection, some minor overlapping may occur and need to be ignored. Typically, a presence detector signals absence with some latency, allowing the user to be detected in some other room. This situation results in a presence detected simultaneously in two different locations, during a short time.

Left-open doors. This rule ensures that an Opening role period does not last more than a maximum duration (3 hours in our setting). Notice that this rule applies to the log of role events and does not consider how assistive applications may react to such situation. For example, HomeAssist includes an application that monitors the entrance door and notifies the user when the door is left open and unattended, for a few minutes (the duration is customized for each user). In practice, when applied to the logs of our field study, this rule detected in almost all cases installation problems. One of the reasons is that the participants were routinized in their activities [10] and did not show any significant decline during the field study.

Presence inclusion. Each room in which interactions need to be detected was equipped with a motion detector. Hence, we generalized the Kitchen presence rule as follows: any interaction at a given location, which is not Presence, must be included in a Presence role at the same location.

Presence intersection. Presence inclusion may be too constrained to apply to some situations. Sometimes, we just need to enforce that Presence and some other interaction have a non-empty intersection when they are located in the same area. Such a rule addresses the home entrance, equipped with a contact sensor for the entrance door and a motion detector for the entrance area. Indeed, whether the user opens the door from outside or inside, Presence and Open role events have a non-empty intersection of time periods.

[1] http://phoenix.inria.fr/research-projects/homeassist

Presence requirement. To check whether a motion detector is still active, even though mispositioned, we introduce the following rule: every role, which is not Presence, must be accompanied by a Presence role event at the same location. This Presence must occur at the same time plus or minus 10 minutes. This rule makes explicit the fact that a motion sensor is always coupled with one or more other sensors in our setting. When this rule is violated, it is likely that the presence detector is not working properly because it is still registered but not emitting any information.

C. Methodology

We collected the log data in the home of 24 participants, aged 80 on average and living alone. Logs covered a period of 9 months. But, due to technical problems (*e.g.,* Internet access, sensor gateway, server), some periods of time had to be ignored; these problems can be directly detected at the platform level. The logs of HomeAssist were further cleaned up by eliminating the non-conformant role events that could be detected by a simple system-heartbeat monitoring. Indeed, every sensor emits a heartbeat signal, whose absence is automatically detected by the lower layers of the platform and signalled as a sensor failure.

We had a plan for each partipant's home and the layout of the sensors. This information was used to diagnose problems when conformance violations occurred in the logs. Another resource used to investigate violatios was the tracking sheets filled by professionals administering questionnaires to each participant during the field study.

D. Experimental results

Our goal in pruning the logs of HomeAssist was to show that our approach could detect anomalies, beyond what simple fault tolerant mechanisms could do.

We defined a model for HomeAssist that allows to raise two main types of anomalies: 1) *permanent non-conformance* refers to a rule systematically violated, indicating some permanent mismatch between the infrastructure and the model; 2) *emerging non-conformance* corresponds to a rule that usually checks, but occasionally fails. Let us now examine instances of these anomalies.

Permanent non-conformance.

In one home, Opening interactions occurred for the kitchen cabinet door but the Presence inclusion rule and the Presence intersection rule always failed. Still, the Presence requirement rule never failed indicating that the cabinet door was opened but was not surrounded a Presence interaction in the kitchen; Presence was detected at unrelated times. The investigation of these situations revealed that this cabinet was not located in the kitchen, but rather in an attendant room, as shown in Fig. 4. Clearly, this constitutes an issue in the model. Consequently, we fixed the model by removing the presence inclusion and presence intersection rules for this installation's model.

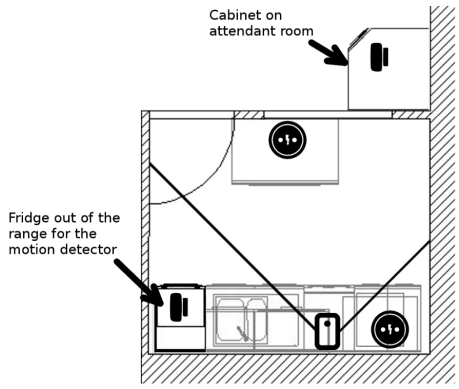

Figure 4. Installation issues

In the same home, another issue was identified: the presence inclusion rule always failed on the fridge opening rule, but the presence intersection rule did not. Even if the fridge was located in the kitchen, the motion detector used to measure presence in the kitchen was not directed so as to cover the entire kitchen, as shown in Fig. 4. This problem is due to an incorrect installation in this case and would simply require the placement of motion sensor to be fixed. However, because we retrofitted HomeAssist in our work, this operation could not be done.

The same situation was observed in two other homes where a kitchen cabinet equipped with a contact sensor were located outside the kitchen.

Typically, these problems arise when assisted living platforms are to be deployed at a large scale. In this context, installations are performed by professionals, not the researchers that designed the platform and possess an implicit knowledge about how to do it right. In our case, even for 24 homes, some installations were performed by non-computer science members of our group that missed some implicit rules.

If our tool had been running during the deployment stage, it would have detected installation anomalies early, that is, during installation or in the first days of operation. Alternatively, as the deployments occurred, additional rules can refine the model to account for unforeseen specificities (*e.g.,* L-shape kitchen). Once the installation or the model has been fixed, the model helps detecting anomalies that did not occur at installation time, but after functioning normally during some time.

Emerging non-conformance.

A pattern of temporary mismatch was observed in the kitchen in five homes at different periods of time. During a few days, the Presence inclusion rule was violated because of an absence of presence in the kitchen due to a faulty motion detector. In all cases, the presence intersection and presence requirement rules were also violated during these periods. The latter rule showed that no presence role was recognized during the 20 minutes surrounding the violation. Results of

theses rules provide helpful information to find the cause of the detected issue. It is thus reasonable to surmise that the cause of the failure is a temporary malfunction of the presence role. This suggests that the motion detector may have been temporarily obstructed or directed incorrectly.

A similar situation occurred for the entrance of two other homes: the door was opened but no presence was detected.

A left-open door was observed in four homes in various locations such as the fridge, the cupboard cabinet, or a wardrobe (*i.e.,* door left open more than three hours). According to the tracking sheets of HomeAssist, the corresponding users were questioned after a few days because of some misbehaviors of applications relying on this interaction. They indicated that the contact sensors felt down. The installations were fixed by the intervention of a member of our group. If our model had been available during the experiment, these incidents would have been reported, and thus fixed, more promptly. This reactivity is a key for context-aware applications. It makes the difference between an application that is useful and one that harasses the user with irrelevant notifications.

VII. Discussion

The previous section showed that an explicit model of an infrastructure is able to detect system malfunctions at installation time or during normal operation. As suggested by some of our examples, once a failure is detected, several diagnosis techniques can be used for identifying the failing role(s).

First, if several rules are violated at the same time, where the rules are checking overlapping sets of sensors, this suggests that the failing role should first be searched in the intersection of these sets. For instance, when the presence intersection rule fails at the same time on the kitchen presence role for different non-presence interaction (fridge, cabinet, *etc.*), this indicates with high probability that the failing role is that of Presence in the kitchen.

Secondly, designing refined versions of a given rule that check a set of sensors or subsets of it, may be useful to direct the search of the failing role or malfunctioning sensor. This idea is illustrated in our model for HomeAssist. There is a chain of three rules R_i expressing increasingly relaxed assertions: presence inclusion, presence intersection, and presence requirement. All these rule are of the type $p \rightarrow q_i$ for $(i = 1..3)$ where the premise p is identical but the conclusion q_i is increasingly weaker. Thus, there is an implication relation along the chain, $R_1 \rightarrow R_2 \rightarrow R_3$, or conversely, when one of the rules fails, stronger rules also do. Based on rule analysis, one can derive a binary decision tree such as the one in Fig. 5 for helping the diagnosis.

When studying some of the systematic mismatches between the installation and a correct model, one may legitimately ask: How come these installation issues were undetected by conventional tests at installation time? For

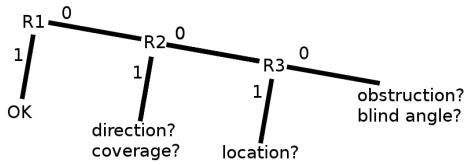

Figure 5. Binary decision tree for helping the diagnosis

instance, the fridge or cabinets placed outside the kitchen would have necessarily failed any test of the breakfast monitoring application. This argument is valid: all *installed* applications were indeed tested in every home at installation time. However, the experimental protocol allowed participants to choose which applications to activate, based on their needs and preferences. As a result, the breakfast monitoring application was not installed in several homes, hence the undetected installation anomalies. If our model had been used at installation time, the installations could have been certified as conform to the model, validating all applications based on this model.

The usefulness of our tool for continuously checking the conformance of an infrastructure goes beyond the context of our field study. First, an infrastructure model provides a reference model for applications available from an online catalog, such as an Appstore: an application can be installed as long as it conforms to the infrastructure model. Secondly, from a practical perspective, checking the model at installation time may reduce the need for testing *all* the deployed application on a given installation, provided that the model completely covers the applications assumptions.

VIII. Related work

Programming models for sensor networks. Much research has been devoted to simplifying the programming of applications over a sensor infrastructure, by defining adequate programming models. Sugihara and Gupta [11] present a comprehensive survey of such approaches. Low-level models simplify the programming of each network node, separately using very small operating systems or virtualization support. High-level models allow to program the whole sensor network as a global system, either 1) using query languages, similar to those used in databases for querying sensor data, 2) writing functional programs to hide the node state manipulations, or 3) extending existing programming languages to address the distributed programming of sensors. While our work also aims to simplify the programming of applications over a network infrastructure, our model of the infrastructure is not intended for programming, but for testing the platform as a global entity. Thus, our infrastructure model is complementary to the programming models of these approaches.

Semantic sensors. Another common approach to abstracting sensor-based applications from details, such as handling failures or dealing with low-level measurements, aims to pro-

vide higher-level sensor abstractions. Semantic Streams [12] allow applications to make semantic queries, such as "detect a vehicle", instead of directly querying low-level magnetometers that exceed a given threshold. The implementation is based on Prolog and allows composing sensors and inference units. Stemming from a software engineering, Software Sensors [13] abstract each hardware sensor as a service implemented in a middleware on top of Jini, a Java-based distributed systems architecture. This approach allows multiple sensors to be combined in a flexible way. There also exist standards for exposing sensors as Web services, such as the Sensor Web Enablement (SWE) standard, promoted by the OGC consortium [14], for improving the interoperability of these services in terms of various aspects (representing measurements, discovering sensors, streaming data from sensors, *etc.*).

These semantic sensor abstractions conceptually correspond to roles in our framework. We also recognize the importance of an abstraction layer over raw sensor measurements and characteristics, building our model at this level of abstraction. In effect, any of the semantic sensor abstractions can be used to implement our notion of role. When retrofitting HomeAssist in our approach, we simulated the semantic sensor layer using Prolog rules, similar to the implementation of Semantic Streams [12], discussed earlier.

Based on the SWE standard cited above, the Semantic Sensor Web [15] adds semantic metadata, leveraging ontologies and inference rules (also standardized by the W3C). For instance, a declarative inference rule, expressed in the Semantic Web Rule Language may define a sensor for blizzard conditions, built on top of sensors for temperature, wind, and precipitation. Our approach also uses declarative inference rules over abstracted sensors (our roles), but with a different goal: verifying the coherence between different sensors data. To the best of our knowledge, this approach has not been used for conformance checking with respect to a model of sensors.

Sensor placement. There has been research efforts to check sensor placement. Hong *et al.* proposed a method based on empirical mode decomposition (EMD) to automatically follow the placement of sensors in the rooms of a building [16], in a context where sensors may be replaced and the physical configuration of the building may evolve. Despite the fact that a home is at a different scale than a building, this method may be helpful to build a model of sensors in an evolving home, but cannot provide information about the conformance between sensors measurements and applications requirements.

Murao *et al.* aim to determine the best position of wearable sensors to monitor activity [17]. Their proposed evaluation function allows to evaluate a sensor position with respect to both its recognition accuracy and its sensor wearability. Because of the constraints they impose, a lot users do not accept wearable sensors to measure daily activities. Surie *et al.* validate a network of sensors by comparing activity performance recognition from a network of sensors with data produced by a wearable camera [18]. Even though it provides information on the infrastructure's reliability, this approach is not scalable because analyzing videos is a time-consuming operation. Furthermore, cameras are viewed as too intrusive to be used in many environments. Research on activity recognition has been done by Philipose *et al.*. Their approach consists to place RFID tags on relevant everyday objects. Thus, interaction recognition is made using an RFID glove [19]. This may avoid sensor positioning issues, but may introduce false positive due to the RFID reception range with respect to the proximity between tagged objects.

Do-it-yourself smart homes. Much research work concerns smart home platforms. The most relevant approaches with respect to our work are those involving the end user in the installation and/or maintenance stages of the system lifecycle. In particular, Kawsar *et al.* propose a framework to support deployment of sensors by the end user in a smart home [20]. Combining this framework with the five design principles to support end-user sensor installation (see Beckmann *et al.* [6]) facilitates the task of a user to position sensors and configure them independently. However, no tools are provided to ensure the appropriate positioning of the sensors and to check their reliability.

Sadoun *et al.* [21] start from natural language descriptions of an installation and its operation scenarios to derive a model of the environment. This model consists of an OWL domain ontology and inference rules in SWRL. Using automatic rule inference, the consistency of the model and the conformance of described operation scenarios with the model can be checked. This checking may signal both incoherent scenarios and incoherence in the installation description itself. The main difference with our work is that their approach only checks the installation description, and not its functioning in the wild. Thereby, their inference rules do not reason about time intervals, which is a key notion for many of our rules, such as no-ubiquity and presence dependency.

IX. Conclusion

We have shown that context-aware applications commonly make implicit assumptions about the sensor infrastructure. These assumptions typically translate into conditional statements that pollute applications code with non-functional concerns. This defensive programming approach can be avoided by expressing these assumptions aside from the application code and factorizing them as an explicit model of the sensor infrastructure. We have expressed this model as a set of rules that can be checked automatically and continuously to ensure the reliability of a sensor infrastructure. Not only does the violation of such rules promptly alerts

about an installation malfunction, but it also contributes to diagnose the problem.

Our approach has been implemented in the context of an assisted living platform, running a set of applications dedicated to assist senior users. Our tool was applied to real sensor data collected during a 9-month field study, consisting of 24 participants aged 80 on average. The results show that some latent installation mistakes could have been found at installation time, using our model. Furthermore, several sensor problems that occurred during operation could have been detected on the fly and repaired more promptly to ensure the reliability of context-awareness applications.

In future work, we will apply our method in a larger deployment consisting in hundreds of installations. This setting will allow us to quantify in more detail the improved reactivity in detecting emerging infrastructure issues, remotely diagnosing the underlying failure, and repairing the platform on-site. Another future work will consist of proposing log visualisation techniques and tools that contribute to identify new rules for the model by recognizing regular event patterns and anomalies.

REFERENCES

[1] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-based activity recognition," *IEEE Transactions on Systems, Man, and Cybernetics*, 2012.

[2] J. Stankovic, I. Lee, A. Mok, and R. Rajkumar, "Opportunities and obligations for physical computing systems," *Computer*, 2005.

[3] J. Bruneau, W. Jouve, and C. Consel, "Diasim: A parameterized simulator for pervasive computing applications," in *Mobile and Ubiquitous Systems: Networking Services, MobiQuitous*, 2009.

[4] W. Edwards and R. Grinter, "At home with ubiquitous computing: Seven challenges," in *Ubicomp 2001: Ubiquitous Computing*, ser. LNCS. Springer, 2001.

[5] K. Henricksen, J. Indulska, and A. Rakotonirainy, "Modeling context information in pervasive computing systems," in *Pervasive Computing*, ser. LNCS. Springer, 2002.

[6] C. Beckmann, S. Consolvo, and A. LaMarca, "Some assembly required: Supporting end-user sensor installation in domestic ubiquitous computing environments," in *UbiComp 2004: Ubiquitous Computing*. Springer, 2004.

[7] J. Durick, T. Robertson, M. Brereton, F. Vetere, and B. Nansen, "Dispelling ageing myths in technology design," in *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, ser. OzCHI '13. ACM, 2013.

[8] L. Caroux, C. Consel, L. Dupuy, and H. Sauzéon, "Verification of daily activities of older adults: a simple, non-intrusive, low-cost approach," in *Proceedings of the ACM SIGACCESS conference on Computers & accessibility*, 2014.

[9] D. Lussier-Desrochers, H. Sauzéon, C. Consel, E. Balland, J. Roux, Y. Lachapelle, V. Godin-Tremblay, and B. N'Kaoua, "Analysis of how people with intellectual disabilities organize information using computerized guidance," *Disability and Rehabilitation: Assistive Technology*, 2016.

[10] V. Bergua, J. Bouisson, J.-F. Dartigues, J. Swendsen, C. Fabrigoule, K. Pérès, and P. Barberger-Gateau, "Restriction in instrumental activities of daily living in older persons: Association with preferences for routines and psychological vulnerability," *The International Journal of Aging and Human Development*, 2013.

[11] R. Sugihara and R. K. Gupta, "Programming models for sensor networks: A survey," *ACM Trans. Sen. Netw.*, 2008.

[12] K. Whitehouse, F. Zhao, and J. Liu, *Wireless Sensor Networks: Third European Workshop, EWSN 2006, Zurich, Switzerland. Proceedings*. Springer, 2006, ch. Semantic Streams: A Framework for Composable Semantic Interpretation of Sensor Data.

[13] E. Lin, *Software Sensors: Design and Implementation of a Programming Model and Middleware for Sensor Networks*. University of California, San Diego, 2004.

[14] M. Botts, G. Percivall, C. Reed, and J. Davidson, *GeoSensor Networks: Second International Conference, GSN 2006*. Springer, 2008, ch. OGC Sensor Web Enablement: Overview and High Level Architecture.

[15] A. Sheth, C. Henson, and S. S. Sahoo, "Semantic sensor web," *IEEE Internet Computing*, 2008.

[16] D. Hong, J. Ortiz, K. Whitehouse, and D. Culler, "Towards automatic spatial verification of sensor placement in buildings," in *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings*, ser. BuildSys'13. ACM, 2013.

[17] K. Murao, H. Mogari, T. Terada, and M. Tsukamoto, "Evaluation function of sensor position for activity recognition considering wearability," in *Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication*, ser. UbiComp '13 Adjunct. ACM, 2013.

[18] D. Surie, O. Laguionie, and T. Pederson, "Wireless sensor networking of everyday objects in a smart home environment," in *Intelligent Sensors, Sensor Networks and Information Processing*. IEEE, 2008.

[19] M. Philipose, K. Fishkin, M. Perkowitz, D. Patterson, D. Fox, H. Kautz, and D. Hahnel, "Inferring activities from interactions with objects," *Pervasive Computing, IEEE*, 2004.

[20] F. Kawsar, T. Nakajima, and K. Fujinami, "Deploy spontaneously: Supporting end-users in building and enhancing a smart home," in *Proceedings of the 10th International Conference on Ubiquitous Computing*. ACM, 2008.

[21] D. Sadoun, C. Dubois, Y. Ghamri-Doudane, and B. Grau, "An ontology for the conceptualization of an intelligent environment and its operation," in *Artificial Intelligence*. IEEE, 2011.